A Proof for P =? NP Problem

WAN ChangLin

The **P** =? **NP** problem is an important problem in contemporary mathematics and theoretical computer science. Many proofs have been proposed to this problem. This paper proposes a theoretic proof for **P** =? **NP** problem. The central idea of this proof is a recursive definition for Turing machine (shortly TM) that accepts the encoding strings of valid TMs. By the definition, an infinite sequence of TM is constructed, and it is proven that the sequence includes all valid TMs. Based on these TMs, the class **D** that includes all decidable languages is defined. By proving **P**=**D**, the result **P**=**NP** is proven.

Categories and Subject Descriptors: F.1.3 [**Complexity Measures and Classes**]: Relations among complexity classes; G.2.1 [**Combinatorics**]: Counting problems

General Terms: Computation, Language, Algorithm

Additional Key Words and Phrases: computational complexity, recursion, decidability, countability

1. INTRODUCTION

The **P** =? **NP** problem is to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some (deterministic) algorithm in polynomial time [Cook 2000]. Since the problem is first posed in Gödel's famous 1956 letter to von Neumann, it has been known as one of the most important problems in contemporary mathematics and theoretical computer science[Sipser 1992]. Gödel considered the **P** =? **NP** problem as a finitary analogue of the Hilbert Entscheidungsproblem: given a mathematical statement, is there a procedure that either finds a proof or finds a disproof.

The notation P stands for "polynomial time". In 1965, Jack Edmonds [Edmond 1965] gave an efficient algorithm to solve the Matching problem and suggested a formal definition of efficient computation (runs in a number of steps bounded by some fixed polynomial of the input size). The class of problems with efficient solutions is later known as **P** class. The **NP** class can be roughly cataloged as a class of problems that have efficiently verifiable solutions. The notation **NP** stands for nondeterministic polynomial time, since originally **NP** was defined in terms of nondeterministic Turing machines [Goldreich 2004]. Nondeterministic Turing machines have more than one possible move from a given configuration. The **P** =? **NP** problem can be intuitively defined as: whether every algorithmic problem with efficiently verifiable solutions have efficiently computable solutions.

2. PROBLEM DEFINITION

In this paper, we mainly follows the official definition announced by Clay Math Institute in 2000. A Turing machine (shortly TM) M is a 4-tuple $\langle \Sigma, \Gamma, Q, \delta \rangle$ where $\Sigma, \Gamma, Q, \delta$ are finite nonempty sets. The set Σ is a finite alphabet with more than one symbol, and the set Γ be a finite alphabet such that $\Gamma \supseteq \Sigma$. The state set Q contains three special states: initial state (q_0), accept state (q_{accept}) and reject

state (q_{reject}). The *transition function* δ satisfies

$$\delta : (Q - \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}.$$

The interpretation of δ is that if M is in state $q \in (Q - \{q_{accept}, q_{reject}\})$ scanning the symbol $s \in \Gamma$ then $q' \in Q$ is the new state, $s' \in \Gamma$ is the symbol printed, and the tape head moves left (-1) or right (1) one square. Let Σ^* be the set of finite strings over Σ, and let Γ^* be the set of finite strings over Γ. A *configuration* of M is a string xqy with $x, y \in \Gamma^*$, y not the empty string, and $q \in Q$. The interpretation of the configuration xqy is that M is in state q scanning the left-most symbol of y, with xy on its tape. A configuration xqy is *halting* if $q \in \{q_{accept}, q_{reject}\}$. For each *nonhalting* configuration C, M has a unique transition configuration C' such that $C \to C'$. The *computation* of M on input $w \in \Sigma^*$ is an unique sequence $C_0, C_1, ...$ of configurations such that $C_0 = q_0 w$ and $C_i \to C_{i+1}$. Therefore, M accepts w iff the computation is finite and the final configuration contains the state q_{accept}. On the contrary, M rejects w iff the computation is infinite or the final configuration contains the state q_{reject}. The number of steps, denoted by $t_M(w)$, is one less than the number of configurations. In terms of TM $M = \langle \Sigma, \Gamma, Q, \delta \rangle$, the elements of the class **P** are languages over Σ. Each language L in **P** is a subset of Σ^*. The language accepted by machine M is defined by

$$L(M) = \{w \in \Sigma^* | M \text{ accepts } w\}.$$

For input size $n \in \mathbb{N}$, the *worst case run time* of M is defined by

$$T_M(n) = max\{t_M(w) | w \in \Sigma^n\}$$

where Σ^n is the set of all strings over Σ of length n. We say that M *runs in polynomial time* if there exists $k \in \mathbb{N}$ such that $T_M(n) \leq n^k + k$ for all n. Therefore, the class **P** of languages is defined by

$$\mathbf{P} = \{L(M) | M \text{ runs in polynomial time}\}.$$

Instead the original nondeterministic definition of **NP**, it can be defined in term of deterministic TM with a *verifying relation*. A verifying relation is a binary relation $R \subseteq \Sigma^* \times \Gamma^*$ for some finite alphabets Σ and Γ. For each verifying relation R, an associate language L_R over $\Sigma \cup \Gamma \cup \{\#\}$ is defined by

$$L_R = \{w \# y | R(w, y)\}$$

where symbol # is not in Σ and Γ. We say that R is *polynomial-time* iff $L_R \in \mathbf{P}$. The class **NP** of languages is defined by

$$\mathbf{NP} = \{L | \forall w \exists y (w \in L \text{ and } |y| \leq |w|^k \text{ and } R(w, y) \text{ is polynomial-time})\}$$

where $|w|$ and $|y|$ denote the lengths of w and y respectively. In other words, a language L over Σ is in **NP** iff there is $k \in \mathbb{N}$ and a polynomial-time verifying relation R over $\Sigma^* \times \Gamma^*$ such that $|w| \leq |y|^k$ for all $w \# y \in L_R$. Finally, with the definitions of **P** and **NP**, the **P** =? **NP** problem is defined by

$$\mathbf{P} = \mathbf{NP}?$$

3. THE CONJECTURE AND APPROACHES TO PROVE IT

3.1 Conjecture

There are two possible answers to the **P** =? **NP** question: **P** = **NP** and **P** ≠ **NP**. Most complexity theorists believe that **P** ≠ **NP**[Cook 2000]. Perhaps this can be partly explained by two facts: first, no **NP** problems are exactly solved in polynomial time so far; second, many results that look like approaching to **P** ≠ **NP** have been achieved by the heavy efforts to prove **P** ≠ **NP**. Although the conjecture **P** ≠ **NP** is still open, the **P** ≠ **NP** assumption plays an important role in modern cryptography [Goldreich 2004]. For instance, the security of the internet, including most electronic transactions, rely on the assumption that it is difficult to factor integer or to break DES (the Data Encryption Standard). The **P** = **NP** conjecture may leads to a much different world. In this world, every problem that has an efficiently verifiable solution, we can find that solution efficiently as well [Fortnow 2009]. Nevertheless, the **P** = **NP** conjecture have some supporting facts as well. Most hard **NP** problems turn to be easy to solve in most cases. For example, Manindra Agrawal, Neeraj Kayal and Nitin Saxena [**?**] proposed a polynomial time algorithm to determinate whether an integer is prime or composite. In fact, there are already practical efficient algorithms, which work well in most cases, for the **NP** problems. Take the SSP (Subset Sum Problem) as an example, Lagarias and Odlyzko [Lagarias and Odlyzko 1985] discovered a density property for SSP and proposed a lattice reduction based method to solve the problem. They proved that their method is efficient to solve low density SSP. Wan and Shi [Wan and Shi 2008] proposed an enumeration based approach for SSP. They proved that their approach is efficient to solve high and median density SSP. The lattice reduction method and enumeration method, with many other approaches, have covered the SSP of almost the whole density scope in two symmetrical directions.

3.2 Approaches

3.2.1 *Completeness.* The serious work on **P** =? **NP** began following the discovery of NP-completeness by Cook [Cook 1971], Karp [Karp 1972], and Levin [Levin 1973] in the seventies. The main results in [Woeginger 2003] are that several natural problems, including SAT and 3-SAT and subgraph isomorphism, are NP-complete, i.e. every problems in **NP** can be reduced to these NP-complete problems in polynomial time. A year later Karp [Karp 1972] shown that 20 other natural problems are NP-complete, and refined the standard notation P, NP and NP-completeness. Independent of Cook and Karp, the notion of "universal search problem", similar to NP-complete problem, is proposed by Leven [Levin 1973] and six examples are given, including SAT. Their work provides a possible way to prove P=NP: an efficient algorithm to any NP-complete problem would imply an efficient algorithm to every NP problem. Woeginger [Woeginger 2003] gave a survey of exact algorithms for **NP** problems.

3.2.2 *Diagonalization.* In 1874 Cantor [Cantor 1874] showed that real numbers are more than algebraic numbers using a technique known as diagonalization. It was used by Gödel in his Incompleteness Theorem, and by Turing in his undecidability results, and then refined to prove computational complexity lower bounds

[Wigderson and Smale 2006]. The diagonalization technique is also used to separate **NP** from **P**. The main idea is that construct an NP language L so that every single polynomial-time algorithm fails to compute L properly on some input. Based on diagonalization, many complexity results are obtained, including Hartmanis and Stearns's Time Hierarchy Theorem, Stearns, Hartmanis and Lewis's Space Hierarchy Theorem, Cook's Nondeterministic Time Hierarchy Theorem and Ladner's Theorem. By the discovery of a feature shared by many similar complexity results, Baker, Gill and Solovay [Baker et al. 1975] suggested the notion relativization and shown that relativizing arguments do not suffice to resolve the P =? NP question.

3.2.3 *Circuit Complexity.* As a model of computation, Boolean circuit is a generalization of Boolean formulae and a rough formalization of the familiar "silicon chip". A Boolean circuit is a diagram showing how to derive an output from an input by a combination of the basic Boolean operations: $OR(\vee)$, $AND(\wedge)$ and $NOT(\neg)$. Because the size of a circuit is the analog of time in algorithms, proving lower bounds for circuits implies lower bounds for algorithms. In 1949, Shannon defined circuit complexity, including monotone circuit complexity, and proved that for almost all Boolean functions $f : \{0,1\}^n \to \{0,1\}$, any Boolean circuit needs at least $2^n/n$ gates to compute f. Based on circuit complexity, a number of new techniques are used to prove circuit lower bounds on natural, restricted classes of circuits, including the Random Restriction method of Furst, Saxe, Sipser [Karchmer and Wigderson 1988] and Ajtai [Ajtai 1983] (proving lower bounds on constant depth circuits), the Communication Complexity method of Karchmer and Wigderson [Furst et al. 1981] (proving lower bounds on monotone formulae). However all attempts in this line fall to obtain any nontrivial lower bounds for general circuits. Razborov and Rudich [Razborov and Rudich 1994] explain this failure by pointing out that any natural proof of a lower bound implies subexponential algorithms for inverting every candidate one-way function, thus natural proofs of general circuit lower bounds are unlikely.

3.2.4 *Proof Complexity.* The concept of proof is a significant property of the study of mathematics. While circuit complexity is used to classify functions according to the difficulty of computing them, proof complexity is used to classify theorems according to the difficulty of proving them. Just like Boolean circuits, proof systems capture the power of reasoning allowed to the prover. By cleverly combing non-relativizing arithmetization with non-naturalizing diagonalization, some proof systems have overcome both the relativization barrier and natural proofs barrier. Buhrman, Fortnow, and Thierauf [Buhrman et al. 1998] first showed that $\mathrm{MA_{EXP}} \not\subset \mathrm{P/poly}$ and proved that their result was non-relativizing by using an oracle A such that $\mathrm{MA}^A_{\mathrm{EXP}} \not\subset \mathrm{P}^A/\mathrm{poly}$. Vinodchandran [Vinodchandran 2005] and Aaronson [Aaronson 2006] showed that for every fixed k, the class **PP** does not have circuits of size n^k and proved that the result was non-relativizing by giving an oracle A such that $\mathrm{PP}^A \subset \mathrm{SIZE}^A(n)$. Santhanam [Santhanam 2007] improved Vinodchandran's result and showed that for every fixed k, the class PromiseMA does not have circuits of size n^k. Aronson and Wigderson [Aaronson and Wigderson 2009] defined the notion algebrization by introducing the algebraic oracles and showed how algebrization captures a new barrier by proving two sets of results. The

first set shows that all known results based on arithmetization that fail to relativize can be obtained by using algebraic oracles. The second set shows that many basic complexity questions, including **P** =? **NP**, will require non-algebrizing techniques.

3.2.5 *Geometric Complexity.* Ketan Mulmuley and Milind Sohoni [Mulmuley and Sohoni 2001] observed that the problems in many complexity classes can be encoded as group actions on vectors in algebraic geometry, and then proposed arithmetic circuit as a new model of computation. Given a family of high-dimension polygons P_n based on group representations on certain algebraic varieties, they argue that for each input of size n, if P_n contains an integral point, then any circuit family for the Hamiltonian path problem must have size at least $n^{\log n}$, which implies $P \neq NP$. They showed that arithmetic circuit bypasses the barriers of relativization, natural proofs, and algebrization by using a natural abstract strategy called "flip" and proposed a new barrier called complexity barrier. Mulmuley [Mulmuley 2009] pointed out that complexity barrier may be the root difficulty of the P =? NP problem by showing the need for the nonelementary techniques to tackle complexity barrier. Although two concrete lower bounds, $P \neq NC$ and $\#P \neq NC$, based on the flip are are obtained, the eventual crossing seems still rely on the solving of Riemman Hypothesis.

4. CONCEPTUAL PROOF

4.1 Decidable Language

Definition 4.1 *Decidable Language.* If a language L can be decided by a TM M, i.e. M accepts any string $w \in L$ and rejects any string $w \notin L$, we say the language L is a decidable language (or L is decidable).

Definition 4.2 *The Class* ***D***. The class **D** is a class of decidable languages. It is defined as

$$\mathbf{D} = \{L(M) | M \text{ is a TM.}\}$$

LEMMA 4.3. *For any finite alphabet* Σ*, the set of all TMs over* Σ *is countable.*

PROOF. For any finite alphabet Σ, we denote the set of all strings over Σ by Σ^*. For each integer $n \geq 0$, the strings of length n are finite, therefore the set Σ^* can be counted in such a way:

Step 0: count all strings of length 0;
Step 1: count all strings of length 1;
...
Step n: count all strings of length n;
...

Because each TM has an encoding into a string $\langle M \rangle$ and Σ^* is countable, the set of all TMs is countable. □

LEMMA 4.4. *For any finite alphabet* Σ*, the class* ***D*** *over* Σ *is countable.*

PROOF. Because the set of all TMs over Σ is countable, the languages $L(M)$ over Σ are countable. From the definition of the class **D**, we have that the class **D** is countable. □

4.2 Recursion and Turing machine

Since the set of all TMs is countable, it is natural to find a method to count all TMs. The main problem of counting all the TMs is how to distinguish those valid encoding strings from all strings over the given alphabet. Based on the theory of recursion, we propose a recursive definition for a TM sequence **Q** to distinguish valid TMs and their encoding strings. The TMs in **Q** take the strings over given alphabet as input and accept those strings which are recognized as valid encoding strings of TMs. For any finite alphabet Σ, the TM sequence **Q** is defined as follows:

M_{00} : {return $reject$.}

M_{10} : {for any input string w, if M_{00} accepts w then
return $accept$
else if $w = \langle M_{00} \rangle$ then
return $accept$
else return $reject$.}

M_{11} : {for any input string w, if M_{00} accepts w then
return $accept$
else if $w = \langle M_{11} \rangle$ then
return $accept$
else return $reject$.}

...

$M_{(i)(2*j)}$: {for any input string w, if $M_{(i-1)(j)}$ accepts w then
return $accept$
else if $w = \langle M_{(i-1)(j)} \rangle$ then
return $accept$
else return $reject$.}

$M_{(i)(2*j+1)}$: {for any input string w, if $M_{(i-1)(j)}$ accepts w then
return $accept$
else if $w = \langle M_{(i)(2*j+1)} \rangle$ then
return $accept$
else return $reject$.}

...

The figure 1 illustrates the recursive definition of all TMs and organizes the TMs in the form of binary tree.

According to the definition, each TM M_{ij} in the sequence **Q** is identified by two integers i and j. Integer i is the number of TMs whose encoding strings are accepted by M_{ij}, integer j is the order number of M_{ij} among all TMs that accepts i TM encoding strings. The first TM M_{00} is the smallest TM that rejects any input string, therefore the encoding $\langle M_{00} \rangle$ of M_{00} should has the shortest size among all TMs over Σ. Except M_{00}, each TM $M_{(i)(2*j)}$ in **Q** accepts all strings that accepted by $M_{(i-1)(j)}$ and the encoding string $\langle M_{(i-1)(j)} \rangle$ of $M_{(i-1)(j)}$, and each TM $M_{(i)(2*j+1)}$ in **Q** accepts all strings accepted by $M_{(i-1)(j)}$ and the encoding string $\langle M_{(i)(2*j+1)}$ of $M_{(i)(2*j+1)}$. Through this recursive way, each TM accepts no more strings than its previous TM and then generate an infinite TM sequence **Q**

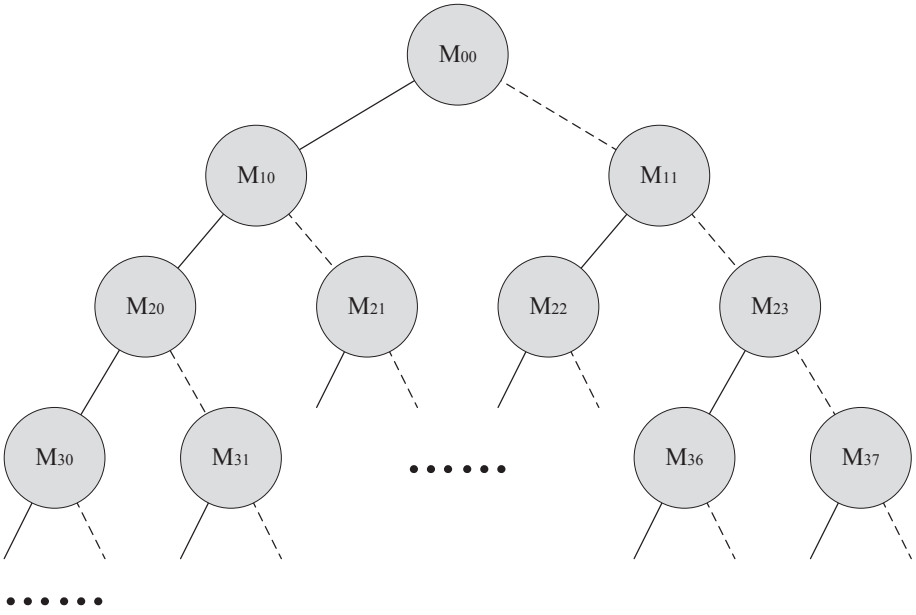

Fig. 1. The recursively defined TM sequence in the form of binary tree.

$= \langle M_{00}, M_{10}, M_{11}, ..., M_{(i)(2*j)}, M_{(i)(2*j+1)}, ... \rangle$.

In the following section, we will show some properties of **Q** and then prove that **P**=**NP**.

4.3 P=NP

LEMMA 4.5. *For any finite alphabet* Σ*, the TM sequence* ***Q*** *includes all valid TMs over* Σ*.*

PROOF. **Induction**

Step 1:

It is obvious that M_{00} is the only TM that accepts none TM encoding string.

Step 2:

Assume there is a TM $M_{1k}, k > 1$, which accepts only one TM encoding string $\langle M \rangle$. The TM M is not M_{00} or itself M. Therefore, TM M at least accepts one different TM M' that is not equal to M. According to the recursive definition, TM M_{1k} can accept the encoding string that is accepted by M. As a result, M_{1k} accepts more than one encoding string. It is contradict to the assumption that M_{1k} accepts only one encoding string. Therefore, there are no TMs accept only one TM encoding string except M_{10} and M_{11} in **Q**.

Step 3:

Suppose all TM M_{ij} that accepts i TM encoding strings are included in the TM sequence **Q**.

Step 4:

Assume there is a TM $M_{(i+1)(k)}$, accepting $i+1$ TM encoding strings, that is not

in the TM sequence **Q**. Because all TM M_{ij} that accepts i TM encoding strings is included in **Q**, there are at most one TM whose encoding string $\langle M \rangle$ is accepted by $M_{(i+1)(k)}$ but not accepted by any M_{ij}. According to the recursive definition of TM in **Q**, the TM M must not be any M_{ij} or $M_{(i+1)k}$. Otherwise, $M_{(i+1)(k)}$ is included in the TM sequence **Q**. Therefore, TM M at least accepts one different TM M' that is not equal to M. As a result, $M_{(i+1)(k)}$ accepts at least $i+2$ TM encoding strings. It is contradict to the assumption that $M_{(i+1)k}$ accepts $i+1$ TM encoding strings. Therefore, there are no TMs accepting $i+1$ TM encoding strings except the TM $M_{(i+1)(j)}$ in **Q**.
Step 5:
By Induction, we proved that the TM sequence **Q** includes all valid TMs over Σ.
□

LEMMA 4.6. ***P*⊆*NP***.

PROOF. The proof of this lemma is trivial. Given any finite alphabet Σ that has at least one element, for each language L over Σ, if $L \in \mathbf{P}$, then there exists at least one polynomial-time verifying relation $R \subseteq \Sigma^* \cup \Sigma^*$ such that $R(w, y) \Leftrightarrow w \in L$ for all $w, y \in \Sigma^*$. Thus we have $\mathrm{P} \subseteq \mathrm{NP}$. □

LEMMA 4.7. ***P*** $\subseteq$ ***D***.

PROOF. According to the definition of class **P**, each language L in **P** can be decided by a TM that runs in polynomial time. Thus L is a decidable language, and then we have $\mathbf{P} \subseteq \mathbf{D}$. □

LEMMA 4.8. *Every TM in **Q** runs in polynomial time.*

PROOF. **Induction**
Step 1:
It is obvious that M_{00} runs in polynomial time because it always halts in one step.
Step 2:
We assume that M_{ij} runs in polynomial.
Step 3:
Because $M_{(i+1)(2*j)}$ and $M_{(i+1)(2*j+1)}$ always simulates M_{ij} first, if the input string w is accepted by M_{ij}, $M_{(i+1)(2*j)}$ and $M_{(i+1)(2*j+1)}$ run like M_{ij} in polynomial time. Otherwise, the input string w is rejected by M_{ij}, and then $M_{(i+1)(2*j)}$ and $M_{(i+1)(2*j+1)}$ run one more step to decide whether w equals to the encoding string of M_{ij} or $M_{(i+1)(2*j+1)}$, i.e. M_{k+1} runs in polynomial time too.
Step 4:
By Induction, we proved that every TM in **Q** runs in polynomial time.
□

LEMMA 4.9. ***D*** $\subseteq$ ***P***.

PROOF. For each language $L(M)$ in **D**, there is a TM M in **Q** that decides L. Because alls TM in **Q** run in polynomial time, we have that $L(M)$ in **P** and $\mathbf{D} \subseteq \mathbf{P}$. □

THEOREM 4.10. ***P*=*NP***

PROOF. Because $\mathbf{D} \subseteq \mathbf{P}$ and $\mathbf{P} \subseteq \mathbf{D}$, we have that $\mathbf{D} = \mathbf{P}$. Suppose $\mathbf{P} \neq \mathbf{NP}$, because $\mathbf{P} \subseteq \mathbf{NP}$, there is at least one language L such that $L \in \mathbf{NP}$ and $L \notin \mathbf{P}$.

Because $\mathbf{P} = \mathbf{D}$, we have a contradiction that the language L is not in $\mathbf{D}$. Therefore, we proved that $\mathbf{P} = \mathbf{NP}$. □

5. CONCLUSION

This paper first give an introduce to the $\mathbf{P}$ =? $\mathbf{NP}$ problem through a survey of related conjecture and approaches to prove it. After a formal definition of decidable language, this paper give a recursive definition for Turing machine that accepts the encoding strings of valid TM. Based on the definition, an infinite sequence of TM is constructed and it is proven that the sequence includes all valid TMs. Based on these TMs, the class $\mathbf{D}$ that includes all decidable languages is defined. At last the result $\mathbf{P}$=$\mathbf{NP}$ is proved by proving $\mathbf{P}$=$\mathbf{D}$.

ACKNOWLEDGMENT

There are many people the author wants to thank. In addition to the people who are mentioned in the paper and references, the author would like to thank Professor Shi Zhongzhi for his kind advisory, thank Professor Donald E. Knuth, one of the scientists I respect most, for his unbelievable program *LaTeX*. This paper may not exist if without *LaTeX*. Thank Professor Chen Xilin, Professor Shui Yuefei, Professor He Qing, Professor Hu Hong, Professor Franco P. Preparata and Professor Yin Jianpin for their helpful advises; thank all reviewers, especially Dr. Petter Strandmark, for their careful review and valuable comments. At last, the author would like to thank the people who give him warm help and encouragement, while it may be not proper to write their noble names at this time.

REFERENCES

Aaronson, S. 2006. Oracles are subtle but not malicious. In *CCC '06: Proceedings of the 21st Annual IEEE Conference on Computational Complexity*. IEEE Computer Society, Washington, DC, USA, 340–354.

Aaronson, S. and Wigderson, A. 2009. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory 1,* 1, 1–54.

Ajtai, M. 1983. σ_1-formulae on finite structures. *Ann. Pure Appl. Logic* 1, 1–48.

Baker, T., Gill, J., and Solovay, R. 1975. Relativizations of the P =? NP question. *SIAM Journal on computing 4*, 431–442.

Buhrman, H., Fortnow, L., and Thierauf, T. 1998. Nonrelativizing separations. *Computational Complexity, Annual IEEE Conference on 0*, 8.

Cantor, G. 1874. Ueber eine eigenschaft des inbegriffes aller reellen algebraischen zahlen. *Crelles Journal f. Mathematik 77*, 258 – 262.

Cook, S. 2000. The p versus np problem.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 151–158.

Edmond, J. 1965. Paths, trees, and flowers. *Canadian Journal of Mathematics 17*, 449–467.

Fortnow, L. 2009. The status of the p versus np problem. *Commun. ACM 52,* 9, 78–86.

Furst, M., Saxe, J. B., and Sipser, M. 1981. Parity, circuits, and the polynomial-time hierarchy. In *SFCS '81: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA, 260–270.

Goldreich, O. 2004. *Foundations of cryptography, volume 1.* Cambridge University Press.

Karchmer, M. and Wigderson, A. 1988. Monotone circuits for connectivity require super-logarithmic depth. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 539–550.

KARP, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 85–103.

LAGARIAS, J. C. AND ODLYZKO, A. M. 1985. Solving low-density subset sum problems. *J. ACM 32,* 1, 229–246.

LEVIN, L. A. 1973. Universal sequential search problems. *Problems of Information Transmission 9,* 3.

MULMULEY, K. 2009. On p vs. np, geometric complexity theory, and the riemann hypothesis. *CoRR abs/0908.1936*.

MULMULEY, K. AND SOHONI, M. A. 2001. Geometric complexity theory i: An approach to the p vs. np and related problems. *SIAM J. Comput. 31,* 2, 496–526.

RAZBOROV, A. A. AND RUDICH, S. 1994. Natural proofs. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 204–213.

SANTHANAM, R. 2007. Circuit lower bounds for merlin-arthur classes. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 275–283.

SIPSER, M. 1992. The history and status of the p versus np question. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 603–618.

VINODCHANDRAN, N. V. 2005. A note on the circuit complexity of pp. *Theor. Comput. Sci. 347,* 1-2, 415–418.

WAN, C. AND SHI, Z. 2008. Solving medium-density subset sum problems in expected polynomial time: An enumeration approach. In *FAW*. 300–310.

WIGDERSON, A. AND SMALE, S. 2006. Np and mathematics - a computational complexity perspective. In *Proc. of the ICM 06*. 665–712.

WOEGINGER, G. J. 2003. Exact algorithms for np-hard problems: a survey. 185–207.